Urządzenia Peryferyjne Laboratorium

21.10.2023

Grupa nr 7B, Czwartek, godz. 11:00, tydzień nieparzysty

Autorzy:

264488 Adam Czekalski

252560 Karol Szydlak

nr	Treść zadania	data wykonania
1.	Ćwiczenie 4 Sterowanie silnikiem krokowym	12/10/2023

Spis treści

1. Wstęp 1
2. Sterowanie 1
3. Aplikacja 2
4. Wykonanie n kroków w lewo lub w prawo w poziomie 5
5. Wykonanie n kroków w lewo lub w prawo w pionie 5

1. Wstęp

W trakcie laboratorium zrobiono aplikację z interfejsem graficznym pozwalającą na:

- Otwarcie i zamknięcie urządzenia za pomocą przycisków
- Wybór sposobu sterowania silnikiem (falowe, pełno-krokowe, pół-krokowe)
- Wykonanie n kroków w prawo oraz n kroków w lewo za pomocą odpowiednich przycisków
- Wprowadzenie liczby kroków oraz czas trwania kroku
- Aplikacja umożliwia sterowanie dwoma silnikami przedstawionymi w instrukcji (S1 oraz S2)

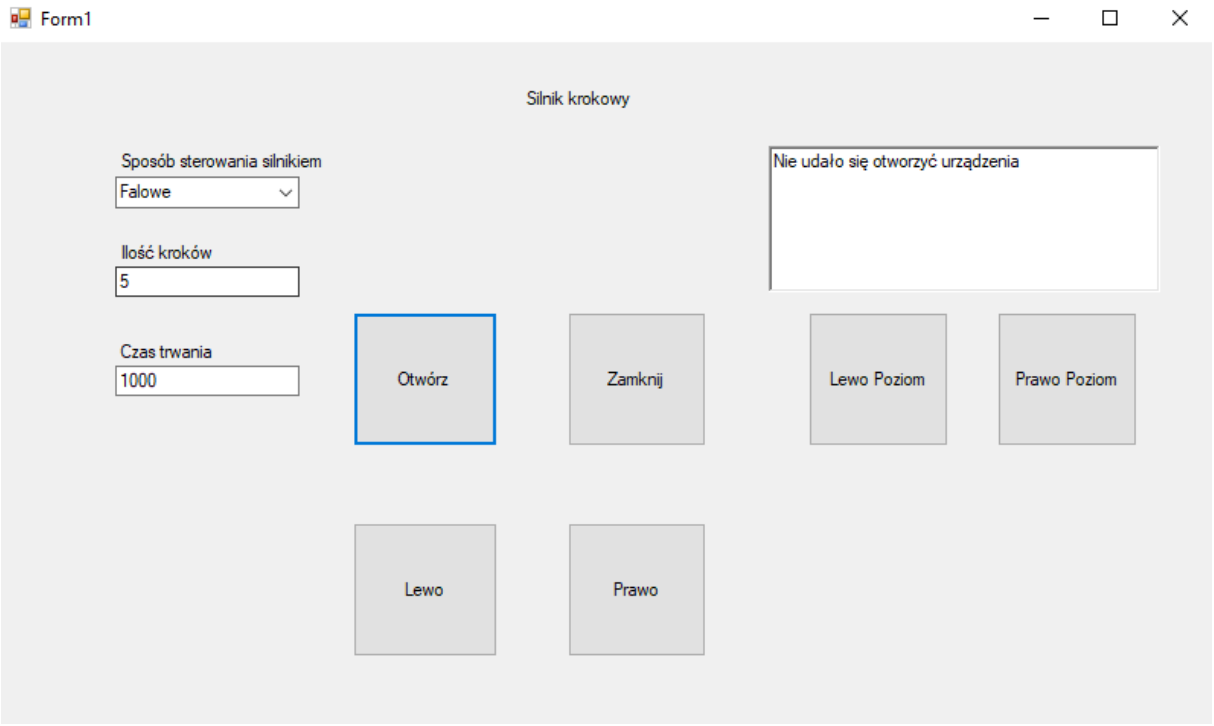
Silnik krokowy jest silnikiem elektrycznym, w którym impulsowe zasilanie prądem powoduje obrót wirnika o ustalony kąt. W laboratorium mieliśmy do czynienia z dwoma silnikami unipolarnymi. Każdy z silników zawiera 6 wyprowadzeń, sterowanie nim polega na przesyłaniu impulsów elektrycznych w odpowiednich uzwojeniach cewek w odpowiedniej kolejności. Do sterowania silnikami użyto płytek: ULN2803 oraz Mmusb245.

2. Sterowanie

- Sterowanie falowe
  Sterowanie falowe jest najprostszym zaimplementowanym sposobem sterowania silnikiem. W starowaniu falowym (zwanym jednofazowym) w danym momencie zasilane jest tylko jedna uzwojenie jednej pary cewek.
- Sterowanie pełno-krokowe
  W sterowaniu pełno-krokowym dwufazowym zasilane są zawsze dwa uzwojenia, po jednym z każdej pary cewek. Przy każdym impulsie generatora zostaje przełączone jedno uzwojenie jednej z par.
- Sterowanie pół-krokowe
  Sterowanie pół-krokowe jest złączeniem dwóch wymienionych wyżej sposobów sterowania. W kolejnych cyklach sterowania zasilamy najpierw jedną fazę, w następnym dwie co powoduje skok o pół-kroku.

3. Aplikacja

Aplikacja do sterowania silnikiem krokowym została stworzona w języku C#, korzystając z biblioteki D2XX oraz z Windows Forms w celu stworzenia interfejsu graficznego.



Rys 1. Interfejs graficzny napisanej aplikacji

3.1 Otwieranie i zamykanie urządzenia

Na początku napisano funkcję umożliwiającą otwarcie i zamknięcie połączenia z silnikiem za pomocą przycisku na interfejsie graficznym.

```
public void openDevice() //otwarcie urządzenia FTDI
{
    richTextBox1.Clear();
    try
    {
        UInt32 ftdiDeviceCount = 0;
        myFtdiDevice = new FTDI();
        myFtdiDevice.GetNumberOfDevices(ref ftdiDeviceCount);
        FTDI.FT_DEVICE_INFO_NODE[] ftdiDeviceList = new FTDI.FT_DEVICE_INFO_NODE[ftdiDeviceCount];
        myFtdiDevice.GetDeviceList(ftdiDeviceList);
        ftStatus = myFtdiDevice.OpenBySerialNumber(ftdiDeviceList[0].SerialNumber);
        myFtdiDevice.SetBitMode(0xff, 1);
        richTextBox1.Text = "Otwarto urządzenie: " + ftdiDeviceList[0].SerialNumber;
    }
    catch
    {
        richTextBox1.Text += "Nie udało się otworzyć urządzenia";
    }

}
```

Rys 2. Funkcja otwierająca urządzenie

Na początku tworzymy nowy obiekt typu urządzenie FTDI, sprawdzamy liczbę urządzeń podłączonych do komputera. Następnie tworzymy strukturę przechowującą informacje o podłączonych urządzeniach, m.in. nr seryjny urządzenia, który potem wykorzystujemy do otwarcia połączenia z urządzeniem poprzez metodę OpenBySerialNumber().

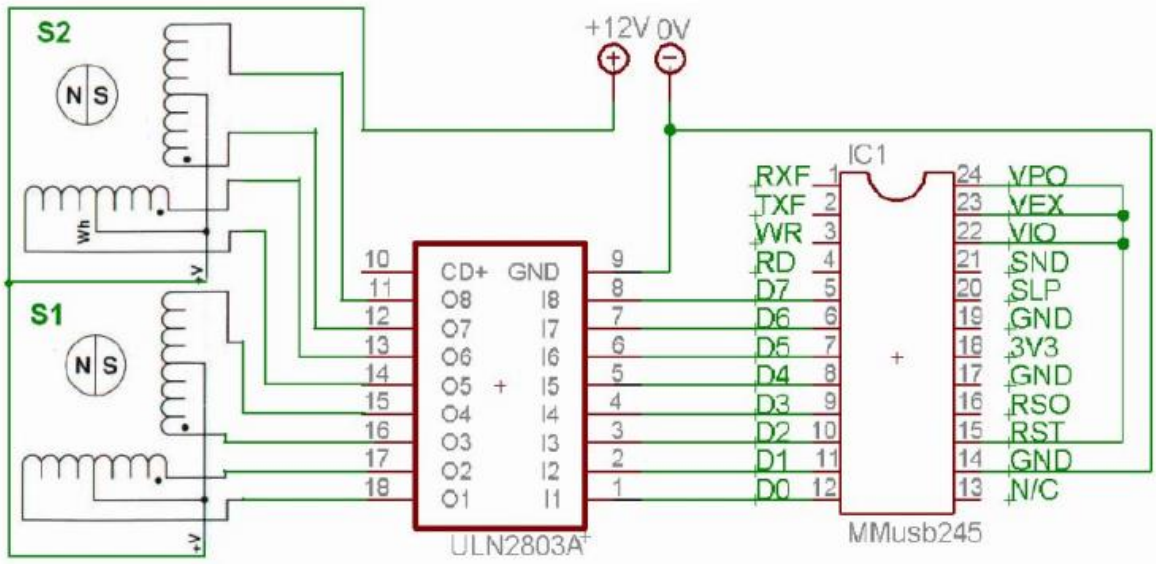
```
public void closeDevice()   //zamknięcie urządzenia FTDI
{
    richTextBox1.Clear();
    if(ftStatus == FTDI.FT_STATUS.FT_OK)
    {
        try
        {
            myFtdiDevice.Close();
        }

        catch
        {
            richTextBox1.Text += "Nie udało się zamknąć urządzenia";
        }
    }
    else
    {
        richTextBox1.Text = "Brak podłączonych urządzeń";
    }
}
```

Rys 3. Funkcja zamykająca urządzenie

Natomiast funkcja zamykania urządzenia opiera się na tym, że najpierw sprawdzamy czy urządzenie jest otwarte, jeśli tak, to wywołujemy metodę Close().

3.2 Wybór sposobu sterowania silnikiem



Rys 4. Schemat układu sterowania silnikami krokowymi S1, S2

Bazując na schemacie z Rysunku 4 oraz wiedzy na temat silników krokowych, stworzono słowa sterujące dla: falowego, pół-krokowego i pełno-krokowego sterowania silnikiem:

```
//Poniższe tablice zawierają sekwencje sterujące silnika krokowego
byte[] fullstepArray = { 0b0110, 0b1010, 0b1001, 0b0101 };  //sterowanie pełnokrokowe silnika S1
byte[] fullstepUpAndDown = { 0b01100000, 0b10100000, 0b10010000, 0b01010000};   // Sterowanie pełnokrokowe silnika S2
byte[] waveArray = { 0b0100, 0b0010, 0b1000, 0b0001 };  //Sterowanie falowe silnika S1
byte[] waveUpAndDown = { 0b01000000, 0b00100000, 0b10000000, 0b00010000 };// Sterowanie falowe silnika S2
byte[] halfstepArray = { 0b0010, 0b1010, 0b1000, 0b1001, 0b0001, 0b0101, 0b0100, 0b0110 };// Sterowanie półkrokowe silnika
byte[] halfstepUpAndDown = { 0b00100000, 0b10100000, 0b10000000, 0b10010000, 0b00010000, 0b01010000, 0b01000000, 0b0110000
```

Rys 5. Sekwencje sterujące

„1” w słowie binarnym oznacza aktywację linii danego portu. 4 najmłodsze bity sterują silnikiem S1, zaś 4 najstarsze silnikiem S2. Zdefiniowano osobne tablice z sekwencjami sterującymi dla silników S1 oraz S2. Przesyłanie kolejnych słów sterujących powoduje pojawienia się stanu wysokiego na odpowiednich portach wyjścia układu sterującego, co powoduje przesłanie impulsu elektrycznego na uzwojenie cewki. Poniżej opisano zasilanie cewek w kolejnych krokach sterowania (jako + rozumie się uzwojenie cewki oznaczone znakiem „•” na Rysunku 3, cewka 1 podłączona jest do O1, O2 urządzenia sterującego natomiast cewka 2 do O3, O4).

W sterowaniu pełno-krokowym:

1) Zasilanie uzwojeń „+” cewek 1 i 2
2) zasilanie uzwojenia „-„ w cewce 2 oraz „+” w cewce 1
3) zasilanie uzwojenia „-„ w cewce 1, oraz 2
4) zasilanie uzwojenia „+” w cewce 2 oraz „-„ w cewce 1
5) powrót do kroku 1.

W sterowaniu falowym:

1) zasilanie „+” uzwojenia cewki 2
2) zasilanie uzwojenia „+” cewki 1
3) zasilanie uzwojenia „-„ cewki 2
4) zasilanie uzwojenia „-„ cewki 1
5) powrót do kroku 1.

W sterowaniu pół-krokowym:

1) zasilanie uzwojenia „+” cewki 1
2) zasilanie uzwojenia „+” cewki 1 oraz „-„ cewki 2
3) zasilanie uzwojenia „-” cewki 2
4) zasilanie uzwojenia „-” cewki 1 oraz „-„ cewki 2
5) zasilanie uzwojenia „-” cewki 1
6) zasilanie uzwojenia „-” cewki 1 oraz „+” cewki 2
7) zasilanie uzwojenia „+” cewki 2
8) zasilanie uzwojenia „+” cewki 2 oraz „-„ cewki 1
9) powrót do kroku 1.

Przy wykonywaniu obrotów w prawo słowa sterujące są podawana od elementu w tablicy o największym indeksie, do tego o najmniejszym (kolejność zasilanie cewek opisana na poprzedniej stronie jest odwrotna).

4. Wykonanie n kroków w lewo lub w prawo w poziomie

```
int index4 = 0; //zmienna służąca do pamiętania indeksu ostatniego przesłanego sygnału sterującego
int index8 = 8; //zmienna służąca do pamiętania indeksu ostatniego przesłanego sygnału sterującego
```

Rys 6. Zmienne służące do pamiętania ostatnio wysłanego słowa sterującego

Tak jak opisane w komentarzu, celem powyższych zmiennych jest zapamiętanie aktualnej pozycji silnika krokowego, żeby w razie kolejnych żądań jego przesunięcia, pamiętał w jakiej pozycji znajdował się wcześniej.

```
// Metoda move służy do przesłania danych na układ MMusb245
// count-> liczba kroków, direction->kierunek, data-> przesyłane słowo sterujące, speed-> czas między krokami
// index-> służy do zapamiętania, które słowo z tablicy sygnałów sterujących zostało przesłane jako ostatnie (aby uniknąć obrotów w przeciwnym niż zamierzonym kierunku)
public void move(int count, int direction, byte[] data, int speed, ref int  index)
{

Int32 bytesToWrite = 1;

    for (int j = 0; j < count; j++)
    {
    index += direction;

    if (direction == -1 && (index <0 || index > data.Length-1))
        {
            index = data.Length-1;
        }
        else
        {
            if (direction == 1 && (index > data.Length - 1 || index < 0))
            {
                index = 0;
            }
        }


    byte[] x = { data[index] };
        myFtdiDevice.Write(x, bytesToWrite, ref numBytesWritten);

        Thread.Sleep(speed);
    }
}
```

Rys 7. Funkcja move

Metoda *move()* steruje silnikiem. Główna pętla jest wykonywana tyle razy ile wynosi zadania ilość kroków. Do zmiennej indeks dodajemy 1, jeśli silnik obraca się w lewo, a odejmujemy 1 gdy silnik obraca się w prawo. Jeśli silnik odczytuje pierwszy element słowa sterującego, podczas obrotu w prawo, indeks musi przeskoczyć na ostatni element słowa sterującego. Natomiast jeśli silnik odczytuje ostatni element słowa sterującego podczas obrotu w lewo, indeks musi przeskoczyć na początek słowa sterującego. Po każdej iteracji następuje wczytanie sekwencji do urządzenia metodą *Write()*. Natomiast uśpienie wątku pozwala na sterowanie prędkością silnika, im większa wartość zmiennej *speed*, tym silnik będzie się wolniej obracał.

5. Wykonanie n kroków w lewo lub w prawo w pionie

Aby urządzenie zaczęło pracować w pionie, wystarczyło podać sygnały na linie silnika S2, co sprowadziło się do przesunięcia słowa bitowego w lewo o 4 bity.

6. Źródła:

[1] https://pl.wikipedia.org/wiki/Silnik_krokowy
[2] https://silniki-krokowe.com.pl/informacje-techniczne/sterowanie-silnikami-krokowymi/
[3] https://silniki-krokowe.com.pl/informacje-techniczne/silniki-krokowe-zasada-dzialania-2/
[4] https://silniki-krokowe.com.pl/informacje-techniczne/sterowanie-polkrokowe-i-mikrokrokowe-silnikow-krokowych/
[5] http://indyk.ict.pwr.wroc.pl/uperyf/cw4.pdf
[6] https://www.monolithicpower.com/stepper-motors-basics-types-uses